Georg-August-Universität
Göttingen
Institut für Informatik

ISSN 1611-1044
Nummer IFI-TB-2006–02

Technischer Bericht

TRex – The Refactoring and Metrics Tool for TTCN-3 Test Specifications

Benjamin Zeiss, Helmut Neukirchen, Jens Grabowski,
Dominic Evans, Paul Baker

Technische Berichte
des Instituts für Informatik
an der Georg-August-Universität Göttingen

3. August 2006

Georg-August-Universität Göttingen
Institut für Informatik

Lotzestraße 16-18
37083 Göttingen
Germany

Tel. +49 (5 51) 39-1 44 14
Fax +49 (5 51) 39-1 44 15
Email office@informatik.uni-goettingen.de
WWW www.ifi.informatik.uni-goettingen.de

TRex – The Refactoring and Metrics Tool for TTCN-3 Test Specifications

Benjamin Zeiss[1]
Helmut Neukirchen[1]
Jens Grabowski[1]
Dominic Evans[2]
Paul Baker[2]

[1] Software Engineering for Distributed Systems Group,
Institute for Informatics, University of Göttingen, Lotzestr. 16-18, D-37083 Göttingen, Germany.
{zeiss,neukirchen,grabowski}@cs.uni-goettingen.de

[2] Motorola Labs, Jays Close, Viables Industrial Estate, Basingstoke, RG22 4PD, UK
{vnsd001,Paul.Baker}@motorola.com

Abstract

The comprehensive test of modern communication systems leads to large and complex test suites which have to be maintained throughout the system life-cycle. Experience with those written in the standardised *Testing and Test Control Notation* (TTCN-3) has shown that the maintenance of test suites is a non-trivial task and its burden can be reduced with appropriate tool support. To this aim, we have developed the TRex tool, published as open-source tool under the Eclipse Public License, which supports the assessment and automatic restructuring of TTCN-3 test suites by providing suitable metrics and refactorings. This paper presents TRex, its functionality, and its implementation.

Keywords

Testing, TTCN-3, Tool, Eclipse, Refactoring, Metrics, Quality Assurance

A short version of this technical report has been published in the proceedings of the TAIC PART (Testing: Academic & Industrial Conference – Practice And Research Techniques) workshop 2006 [1].

1 Introduction

The *Testing and Test Control Notation* (TTCN-3) [6, 12] is a test specification and test implementation language standardised by the *European Telecommunications Standards Institute* (ETSI) and the *International Telecommunication Union* (ITU). While TTCN-3 has its roots in functional black-box testing of telecommunication systems, it is nowadays also used in other domains such as Internet protocols, automotive, aerospace, or service-oriented architectures. TTCN-3 can be used not only for specifying and implementing functional tests, but also for scalability, robustness or stress tests. TTCN-3 is based on a textual core notation and several presentation formats [7, 8].
Commercial TTCN-3 tools [14, 17, 18] and in-house solutions support editing test suites and compiling them into executable code. By implementing the interfaces of the standardised TTCN-3 runtime environment [9, 10], these tools also allow TTCN-3 test campaigns to be managed and executed.
Experience within Motorola has shown that not only editing and executing TTCN-3 test suites, but also maintenance of TTCN-3 test suites is an important issue which requires tool support [2]. For example, the conversion of a legacy test suite for a UMTS based component to TTCN-3 resulted in 60,000 lines of code which were hard to read, hard to (re-)use, and hard to maintain.
Currently, no tools for assessing and improving the quality of TTCN-3 test suites exist. To this end, Motorola has collaborated with the University of Göttingen to develop a TTCN-3 refactoring and metrics tool, called *TRex*. The initial aims of TRex were to: (1) enable the assessment of a TTCN-3 test suite with respect to lessons learnt from experience, (2) provide a means of detecting opportunities to avoid any issues, and (3) a means for restructuring TTCN-3 test suites to improve them with respect to any existing issues. To let others participate in our tool and to participate in contributions from others, we have made TRex a general open-source quality assurance and quality improvement tool for TTCN-3 test suites.
This paper is structured as follows: In the next section, we will give an overview of TRex's functionality following with a description of the implementation in Section 3. In Section 4 we explain the open-source availability of TRex before concluding with a summary and outlook.

2 Functionality of the TRex tool

TRex is implemented as an Eclipse plug-in and therefore, everyone who has experience with the Eclipse Platform [5], e.g. by using the popular *Java Development Tools* (JDT), will immediately feel comfortable with TRex. The TTCN-3 perspective of TRex (Figure 1) allows editing of TTCN-3 core notation as well as assessing and improving the quality of TTCN-3 test suites.

2.1 Editing

The individual files of a TTCN-3 test suite are organised into *projects*. The *Navigator* view (left hand side of Figure 1) allows these projects to be explored and files to be

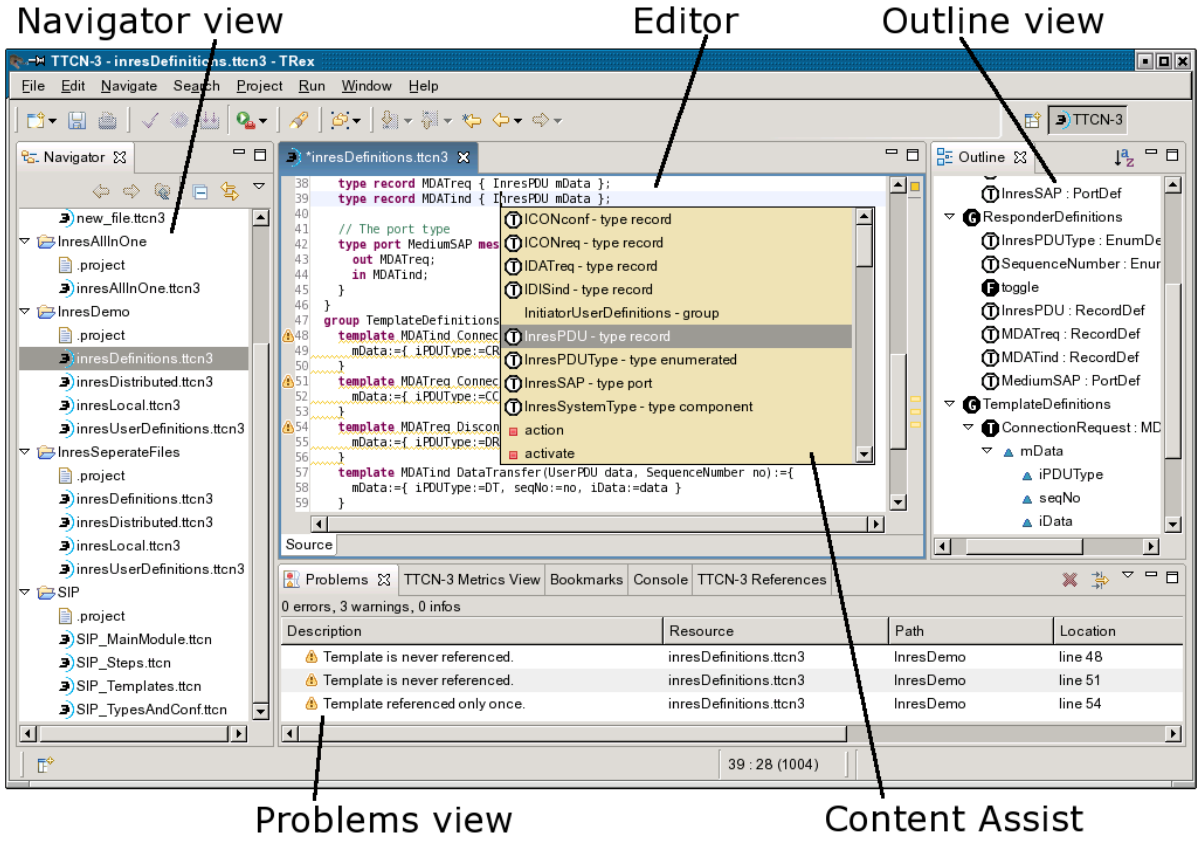


Figure 1: The TRex TTCN-3 perspective

selected for editing. TRex provides an editor (centre of Figure 1) with syntax highlighting and syntax checking[1] which is performed on-the-fly while typing. Syntax errors are listed in the *Problems* view (bottom of Figure 1) and as error markers within the editor window itself. In addition, the *Outline* view (right hand side of Figure 1) provides a tree representation of the TTCN-3 structure for the currently edited file and supports navigation by double-clicking on its elements.

To facilitate typing of descriptive but long identifiers, a *Content Assist* functionality (centre of Figure 1) can be used. Content Assist makes suggestions to complete identifiers based on the typed prefix which is used to match identifiers declared in the test suite. Further assistance on identifiers is provided by a *Text Hover*[2] which is activated if the mouse pointer is moved over an identifier: in this case, type and scope information for that identifier is displayed. If this information is not sufficient, TRex provides the *Open Declaration*[2] functionality to jump to the location where the corresponding element is declared. To identify all locations where a certain element is referenced, the *Find References*[2] functionality can be used; all found referencing locations for the given element being displayed in the *TTCN-3 References* view.

Finally, to ease proper formatting of TTCN-3 core notation, a *Formatter* may be run to beautify existing TTCN-3 files and can be configured to use different coding styles.

[1] TRex supports the latest available version of the TTCN-3 core language specification (v3.1.1).

[2] For *Text Hover*, *Open Declaration*, and *Find References*, no screenshot is provided.

2.2 Compiler integration

To allow the edited tests to be compiled and run either against a real or an emulated system under test, it is necessary to provide integration between TRex and a suitable TTCN-3 to target language (C, Java, etc.) compiler. In order to provide a consistent user experience, the chosen external compiler simply has to support commandline usage and provide a defined and regular format for any error messages that might arise. Currently, we have included support for invoking the Telelogic Tau G2/Tester [17] analyser and compiler *t3cg* from within the TTCN-3 perspective. Warnings and errors generated by this process are collated into the *Problems* view whilst all generated files appear in the *Navigator* view (see Figure 1 for default view locations).

2.3 Refactoring

As a powerful means for improving the quality of TTCN-3 test suites, TRex is able to restructure test suites in an automated way. This is achieved by using *refactoring* which is defined as "*a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior*" [11]. While refactoring is well known for implementation languages like Java, it has not been systematically studied for TTCN-3. Thus, we developed a catalogue of 49 refactorings applicable for TTCN-3 [20, 21]. In TRex, we have begun implementing those refactorings which we believe would improve the maintainability of Motorola's test suites and have so far completed the *Inline Template*, *Inline Template Parameter*, and *Rename* refactorings. As an example, we will describe the *Inline Template* refactoring in detail.
For specifying test data, TTCN-3 uses so called *templates*. A template may either be defined as a named entity on its own or "on-the-fly" using an *inline template* notation. The first way promotes re-use since a template definition may be referenced at several locations. In contrast, test behaviour may be easier to understand if it uses inline templates, since inline templates define test data at the same location where it is actually used for sending or receiving.
The *Inline Template* refactoring allows a template reference to be transformed into its semantically equivalent inline template notation. The application of this refactoring is particularly reasonable if a template is only referenced once. When applying this refac-

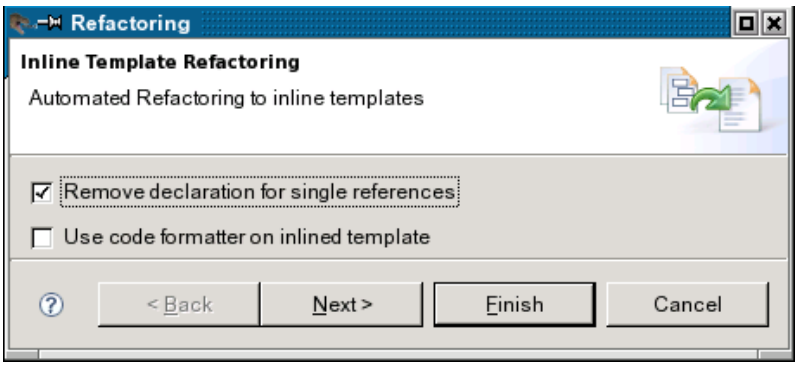


Figure 2: Wizard for the configuration of the *Inline Template* refactoring

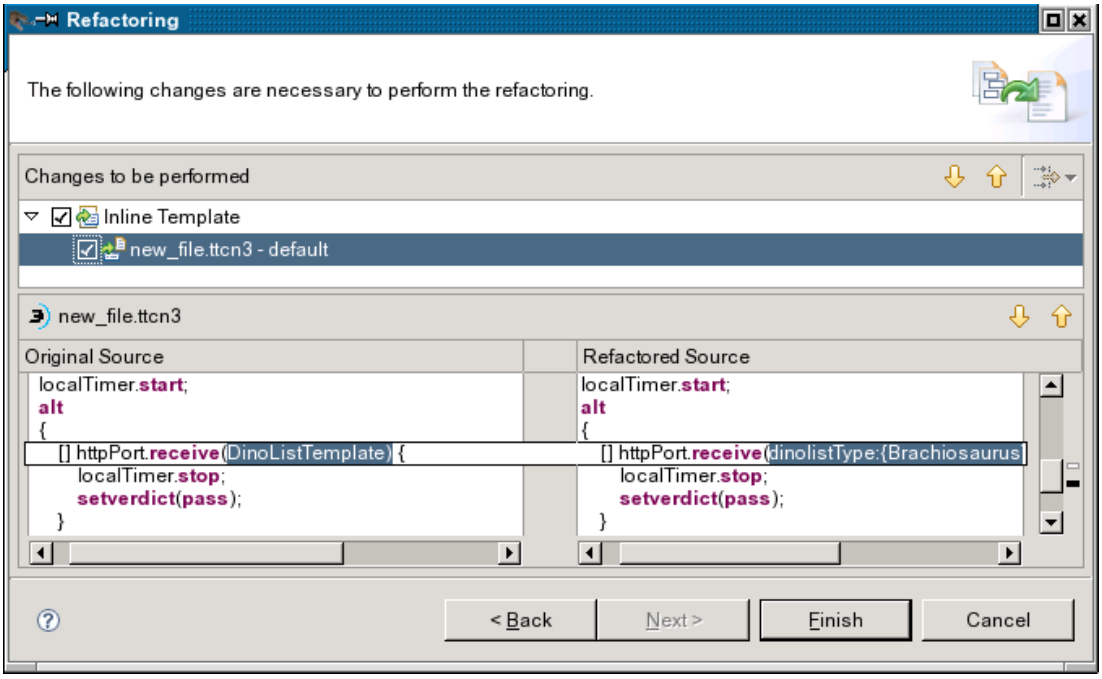
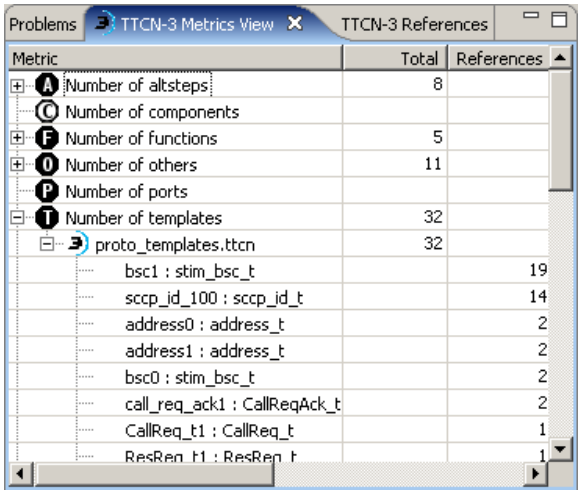


Figure 3: Wizard for the preview of the *Inline Template* refactoring

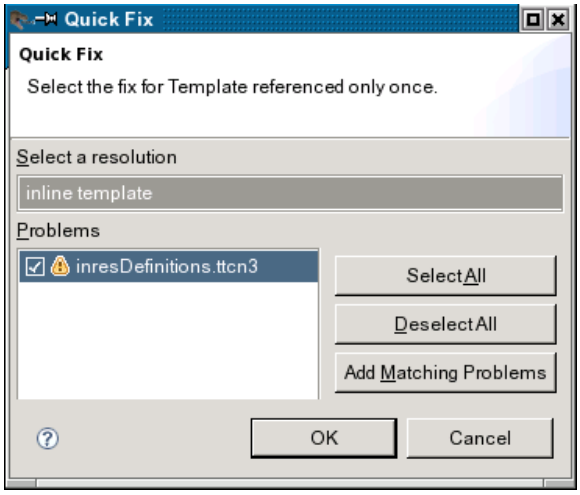
toring to a template reference, TRex opens a wizard dialogue which offers configuration for the *Inline Template* refactoring. As shown in Figure 2, it is possible to remove the declaration of a template if it was referenced only once and the Formatter may additionally be used to obtain a pretty-printed template. Before a refactoring is actually applied, the refactoring wizard displays a preview of all resulting changes (Figure 3).
These refactorings are typically semi-automated, since the user still has to identify locations where they should be applied (as known from JDT for example). However, as shown in the next section, TRex can also automatically identify such locations.

2.4 Metrics

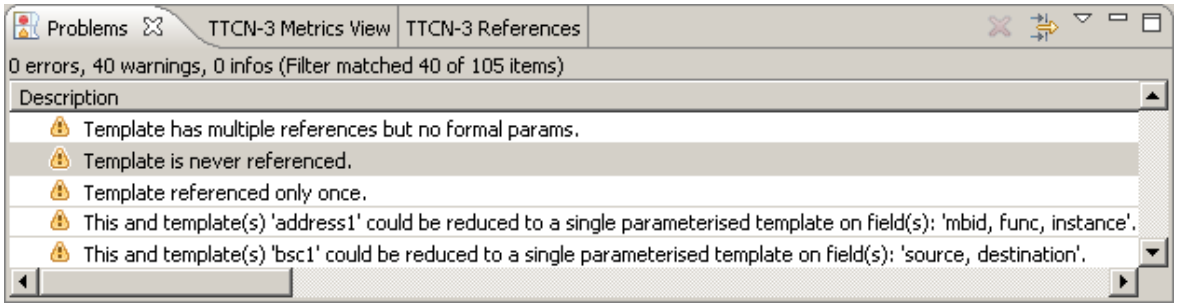
As part of TRex we are investigating the application of metrics to give an indication of both the overall quality of a TTCN-3 test suite and any locations where it might be beneficial to apply a particular refactoring. For this we have defined several TTCN-3 specific examples, based on well-known software metrics, which will be briefly outlined below (full description is available in our previous paper on TTCN-3 refactoring [21]).
We have implemented basic linguistic metrics in the TRex tool, including *Number of . . .* and *References to . . .* for various definitions and types as well as a measure, labelled *Template coupling*, of the dependency between test behaviour and test data (in the form of template definitions). Figure 4(a) shows the *TTCN-3 Metrics* view.
Based on these metrics, we have defined several rules [21] by which the templates of a TTCN-3 test suite can be analysed, e.g. looking at number of references, use of parameters, commonalities, etc. From these, TRex is able to identify problematic code fragments and to suggest suitable refactorings. For example, templates which could be removed, inlined, or merged into a common parametrised version. These suggestions



(a) TTCN-3 Metrics view



(b) TRex's Quick Fix suggestion



(c) TRex's rule-based refactoring suggestions

Figure 4: TRex's Metrics-based functionality

are displayed in the *Problems* view as warnings (Figure 4(c)) and can either be treated merely as indicators that should be taken into account whilst working on the test suite, or an associated *Quick Fix* can be invoked via the context menu to perform a suggested refactoring automatically (Figure 4(b)).

3 Implementation of the TRex tool

TRex is implemented in Java as a set of plug-ins for the Eclipse Platform [5]. Building on Eclipse is attractive from the developer's point of view as it is well documented and supported, and provides many ready-to-use components for the implementation of an *Integrated Development Environment* (IDE). Such components include project and file management (workspace) and a graphical user interface (workbench) which can be configured to match the typical layout of an IDE. In fact, the majority of TRex's functionality is built upon abstract implementations provided by Eclipse.

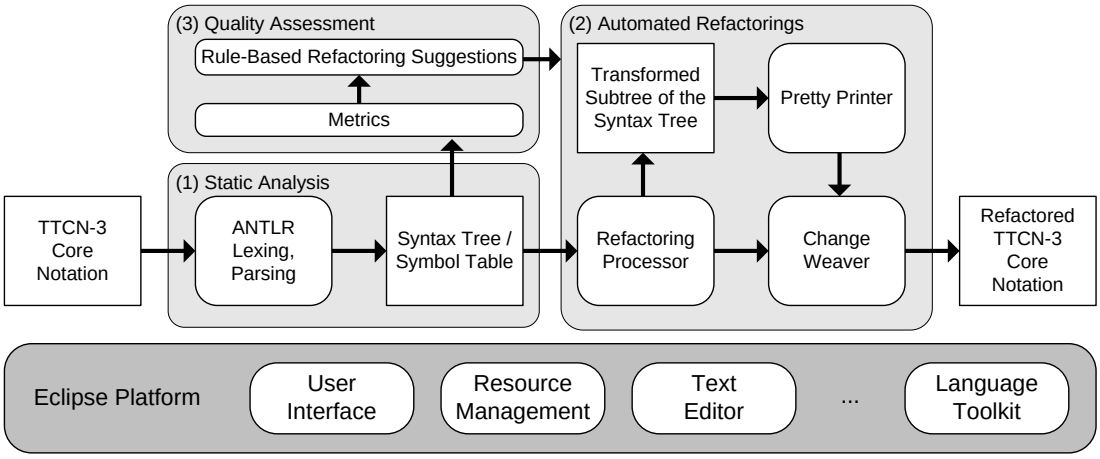


Figure 5: The TRex tool chain

Figure 5 shows the TRex tool chain: the Eclipse Platform provides the basic IDE infrastructure. The TRex components build on top of the Eclipse Platform. They are explained in the subsequent sections.

3.1 Static analysis

The foundation for most functionality in TRex is the TTCN-3 parser and the resulting syntax tree[3]. For building up the syntax tree for a test suite we use *'ANother Tool for Language Recognition'* (ANTLR) [15], a parser generator which supports lexing, parsing, and syntax tree creation and traversal. For tree traversal, ANTLR uses tree grammars (e.g. the pretty printer uses a tree grammar enriched with semantic actions for the syntax reconstruction).

Most of the advanced functionality of TRex requires additional information for TTCN-3 identifiers, such as the identifier's type, or the syntax tree node of its declaration. To easily find this information, a symbol table was implemented. Its underlying data structure is a red-black tree. Although symbol tables commonly use hash tables, the red-black tree provides a more efficient way to implement the Content Assist feature while retaining a generally good performance. As TRex uses multiple passes, the symbol table is stored statically. Each symbol table is part of its scope and the scopes are again organised as a tree.

The syntax tree and the symbol table provide the basis upon which most of TRex's present functionality is implemented, e.g. the metrics and refactoring implementations both use them. For the quality assessment and the automated refactorings, the source files in TTCN-3 core notation are analysed first (Block (1) in Figure 5). In this step, the lexer creates a token stream which is used by the parser for syntactical validation and for building the syntax tree. In addition, the symbol table is also created here.

[3] An alternative approach would be to build up a TTCN-3 meta model [16] representation of a TTCN-3 test suite and to use this representation instead of the syntax tree.

3.2 The refactoring implementation in general

The refactoring implementations make use of the Eclipse *Language Toolkit* (LTK) which provides abstract classes for semantic preserving workspace transformations and customisable wizard pages for the user interaction. The benefit of such wizard pages is, for example, an integrated preview pane that can be used to compare the original source to the refactored source side by side. Most automated refactorings in TRex currently involve the selection of the concerned identifiers (e.g. the identifier to be renamed in the case of the *Rename* refactoring) or code parts in the editor. Therefore, a data structure (called *rangemap*) is needed to find the syntax tree nodes which are associated to editor cursor positions or selections. In TRex, every interaction between the syntax tree, the symbol table and the editor is based on text offset positions rather than lines and columns. This is because Eclipse interprets tab stops differently in offsets and columns. Each identifier which should be stored in the rangemap data structure is identified by a range: a start offset and an end offset. The data structure chosen for the rangemap is again a red-black tree and the offset ranges are used as keys.
Block (2) in Figure 5 depicts how the automated refactorings are realised. On the basis of the static analysis step (Block (1)), the workspace transformations can be calculated once the concerned syntax tree node (or nodes respectively) has been found through the identifier rangemap and the user entered any required information in the refactoring wizard.
The transformation of the workspace resources (i.e. text files) is realised with a programmatic text editor provided by the Eclipse Platform. It supports copy, paste, move, delete, insert, and replace operations. These operations are used to weave only the textually changed parts into the original TTCN-3 source files. Therefore most of the original formatting is preserved. In some cases an intermediate step involving a syntax tree transformation may become necessary, in order to calculate the required changes. In this case, the TTCN-3 core notation to be weaved into the original TTCN-3 source files is obtained by the pretty printer. Applying multiple changes to a single file is supported by the programmatic editor by automatically tracking changing offset positions.

3.3 Individual TTCN-3 refactorings

The *Rename* refactoring is based on a reference finder algorithm which traverses all visible syntax subtrees of a symbol declaration and uses the symbol table to detect whether every reference found with the same name also has the same declaration. This algorithm is highly reusable and is also an essential part of the *TTCN-3 References* view, the *Inline Template* refactoring and rule-based refactoring suggestions.
The *Inline Template* refactoring uses the symbol table to obtain the declaration of a template reference and the pretty printer to generate a template body for the inline template. This template body is rewritten into a syntactically correct inline template which means that its type description is added. Templates using the TTCN-3 *modifies* concept require a different inline template syntax which is also supported by TRex. In the case of parametrised templates, the template parameters are inlined into the template body first. Since inlining of template parameters is also required for the *Inline Template Parameter* refactoring, the implementation of this refactoring works in a similar way.

3.4 Metrics and rule-based refactoring suggestions

Metrics are measured immediately after the syntax tree for a test suite has been built or updated (Block (3) in Figure 5). The tree is then fully traversed; all definitions that metrics will be calculated for (e.g. *altstep*, *function*, *template*, etc.) are recorded and all communication statements (*send* and *receive*) are processed to derive *Template coupling* scorings. References to all of these are calculated in a further pass of the tree, hence giving enough information for the basic linguistic metrics (described in Section 2.4) to be displayed. Then all templates found in the first step are processed one-by-one against the analysis rules, using the previously calculated referencing information as well as further inspection of their structure.
Once this has completed, the rule findings are associated with the templates in the form of customised Eclipse *marker* objects which are automatically displayed in the *Problems* view. *Quick Fixes* are resolved for each of them based on extended attributes which indicate the detected situation and hence some corresponding suggestion(s) from the rule set.

4 Open source availability

TRex is released under the *Eclipse Public License* (EPL) [4] and is publicly available at its website [19]. Since most TTCN-3 tools are either commercial or in-house solutions, we hope that TRex makes TTCN-3 more popular (e.g. for educational purposes).
In addition to the University of Göttingen, Motorola UK has already contributed huge development efforts into TRex. We invite interested Java programmers to contribute as well to the development of TRex. Due to the nature of the EPL, the development of in-house extensions or commercial third-party plug-ins for TRex is possible as well. For such purposes, further TRex specific extension points can be introduced.

5 Summary and outlook

We presented TRex, a general open-source quality assessment and quality improvement tool for TTCN-3 test suites. TRex is implemented as an Eclipse plug-in and provides metrics and refactoring for TTCN-3. Furthermore, special rules, which interpret metric values, support an automatic refactoring of TTCN-3 test suites.
Future versions of TRex will include enhanced editing functionality and further metrics, refactoring, and analyses for TTCN-3 test suites. Therefore, we have started to implement control-flow- and call-graphs for TTCN-3 behaviour [22]. These graphs will be used, for example, to provide complexity metrics and to allow the detection of anomalies in the control- and data-flow.
Besides these improvements to the existing functionality, we plan to add two major components to TRex:
Firstly, the use of metrics to assess the quality of TTCN-3 test suites and to suggest appropriate refactorings is only one possible approach. A further approach, which we would like to pursue, is to identify anti-patterns [3], i.e. inappropriate usage of TTCN-3

(so called “bad smells”). In contrast to the calculation of metrics, this requires a pattern-based approach to be implemented, e.g. to identify duplicate code.
Secondly, in contrast to the application of automated refactorings, manual refactoring of tests suites is error prone. To additionally support manual refactoring of test suites, we would like to add a component which allows the observable behaviour of a test suite to be verified as being the same before and after a refactoring is performed. This will be done by implementing a bi-simulation [13] tool, which executes and compares the two variants of the test suite in parallel.
In addition to the tool development, we have also started to analyse existing real-world TTCN-3 test suites in order to determine appropriate boundary values for our metrics. Without these, the interpretation of metric findings may be very difficult.
TRex is an open-source tool and freely available under the Eclipse Public License. We invite the TTCN-3 community to use the tool, share their experience, and participate in the future development of TRex.

6 Bibliography

[1] P. Baker, D. Evans, J. Grabowski, H. Neukirchen, and B. Zeiß. TRex – The Refactoring and Metrics Tool for TTCN-3 Test Specifications. In *Proceedings of TAIC PART 2006 (Testing: Academic & Industrial Conference – Practice And Research Techniques), Cumberland Lodge, Windsor Great Park, UK, 29th-31st August 2006.* IEEE Computer Society, August 2006.

[2] P. Baker, S. Loh, and F. Weil. Model-Driven Engineering in a Large Industrial Context – Motorola Case Study. In L. Briand and C. Williams, editors, *Model Driven Engineering Languages and Systems: 8th International Conference, MoDELS 2005, Montego Bay, Jamaica, October 2-7, 2005*, volume 3713 of *Lecture Notes in Computer Science (LNCS)*, pages 476–491. Springer, May 2005.

[3] W.J. Brown, R.C. Malveau, and H. McCormick. *Anti-Patterns*. Wiley, 1998.

[4] Eclipse Foundation. Eclipse Public License – Version 1.0. `www.eclipse.org/legal/epl-v10.html`.

[5] Eclipse Foundation. Eclipse. `http://www.eclipse.org`, 2006.

[6] ETSI European Standard (ES) 201 873-1 V3.1.1 (2005-06): The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language. European Telecommunications Standards Institute (ETSI), Sophia-Antipolis, France, also published as ITU-T Recommendation Z.140, 2005.

[7] ETSI European Standard (ES) 201 873-2 V3.1.1 (2005-06): The Testing and Test Control Notation version 3; Part 2: TTCN-3 Tabular Presentation Format (TFT). European Telecommunications Standards Institute (ETSI), Sophia-Antipolis, France, also published as ITU-T Recommendation Z.141, 2005.

[8] ETSI European Standard (ES) 201 873-3 V3.1.1 (2005-06): The Testing and Test Control Notation version 3; Part 3: Graphical Presentation Format for TTCN-3 (GFT). European Telecommunications Standards Institute (ETSI), Sophia-Antipolis, France, also published as ITU-T Recommendation Z.142, 2005.

[9] ETSI European Standard (ES) 201 873-5 V3.1.1 (2005-06): The Testing and Test Control Notation version 3; Part 5: TTCN-3 Runtime Interface (TRI). European Telecommunications Standards Institute (ETSI), Sophia-Antipolis, France, also published as ITU-T Recommendation Z.144, 2005.

[10] ETSI European Standard (ES) 201 873-6 V3.1.1 (2005-06): The Testing and Test Control Notation version 3; Part 6: TTCN-3 Control Interface (TCI). European Telecommunications Standards Institute (ETSI), Sophia-Antipolis, France, also published as ITU-T Recommendation Z.145, 2005.

[11] M. Fowler. *Refactoring – Improving the Design of Existing Code*. Addison-Wesley, 1999.

[12] J. Grabowski, D. Hogrefe, G. Réthy, I. Schieferdecker, A. Wiles, and C. Willcock. An Introduction into the Testing and Test Control Notation (TTCN-3). *Computer Networks*, 42(3), June 2003.

[13] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science (LNCS)*. Springer, 1980.

[14] OpenTTCN Oy OpenTTCN Tester for TTCN-3. `http://www.openttcn.com/Sections/Products/OpenTTCN3`, 2006.

[15] T. Parr. ANTLR parser generator. `http://www.antlr.org`, 2006.

[16] I. Schieferdecker and G. Din. A Meta-model for TTCN-3. In M. Núñez, Z. Maamar, F.L. Pelayo, K. Pousttchi, and F. Rubio, editors, *Applying Formal Methods: Testing, Performance and M/ECommerce, FORTE 2004 Workshops, Toledo, Spain, October 1-2, 2004*, volume 3236 of *Lecture Notes in Computer Science (LNCS)*, pages 366–379. Springer, 2004.

[17] Telelogic Tau/Tester. `http://www.telelogic.com/corp/products/tau/tester/index.cfm`, 2006.

[18] Testing Technologies TTworkbench. `http://www.testingtech.de/products/ttwb_intro.php`, 2006.

[19] TRex Website. `http://www.trex.informatik.uni-goettingen.de`, 2006.

[20] B. Zeiss. A Refactoring Tool for TTCN-3. Master's thesis, Institute for Informatics, University of Göttingen, Germany, ZFI-BM-2006-05, March 2006.

[21] B. Zeiss, H. Neukirchen, J. Grabowski, D. Evans, and P. Baker. Refactoring for TTCN-3 Test Suites. In *Proceedings of SAM'06: Fifth Workshop on System Analysis and Modelling, May 31–June 2, 2006, University of Kaiserslautern, Germany*,

2006. (Extended version to appear as *Refactoring and Metrics for TTCN-3 Test Suites* in the Lecture Notes in Computer Science (LNCS) series published by Springer).

[22] B. Zeiß, H. Neukirchen, J. Grabowski, D. Evans, and P. Baker. TRex – An Open-Source Tool for Quality Assurance of TTCN-3 Test Suites. In *Proceeedings of CONQUEST 2006 – 9th International Conference on Quality Engineering in Software Technology, September 27–29, Berlin, Germany*. dpunkt.Verlag, Heidelberg, September 2006.

7 Authors' biographies

Benjamin Zeiss

Benjamin Zeiss is a doctoral student (Siemens scholarship) at the Software Engineering for Distributed Systems group at the Institute for Informatics at the Georg-August University of Göttingen. His research interests are refactoring and quality assessment of functional tests for reactive systems. He holds a Master of Science degree in Applied Computer Science.

Helmut Neukirchen

Dr. Helmut Neukirchen works at the Software Engineering for Distributed Systems group at the Institute for Informatics at the Georg-August University of Göttingen. He defended his PhD thesis on "Languages, Tools and Patterns for the Specification of Distributed Real-Time Tests". His research combines agile methods and test specification. Helmut Neukirchen is heading the ETSI work item "Patterns in Test Development".

Jens Grabowski

Prof. Dr. Jens Grabowski works at Institute for Informatics at the Georg-August University of Göttingen, where he is head of a research group on software engineering for distributed systems. His research interests include automatic test generation, test specification, test languages and test methodology. Jens Grabowski is an active member in the standardisation of TTCN-3 by ETSI and of the UML 2.0 Testing Profile by OMG.

Dominic Evans

Dominic Evans joined the Motorola UK Research Lab as a contractor in July 2005 and has contributed to several research projects since this time. He was awarded the degree of Master of Engineering in Computer Engineering with first-class honours from the University of Southampton and has experience with numerous programming tools and techniques. His current interests include software quality assurance and model-driven engineering.

Paul Baker

Paul Baker is the manager of Motorola's European System and Software Engineering Research laboratory. He completed his MSc in Software Engineering at the University of Oxford during which he studied formal semantics for scenario-based specification languages and automatic test generation algorithms. Since working at Motorola, Paul has been involved in model-driven development techniques, such as automatic code and test generation and more recently advanced techniques for system and software validation and verification. To support this work he has been involved in the standardisation of Message Sequence Charts (ITU Z.120), the graphical format for TTCN-3, and the OMG's UML 2.0 Test Profile. Paul has a number of patents in the area of test generation, and the analysis of partial specifications.